intel

ARTICLE REPRINT

AR-182

September 1981

Multiprotocol Chip Brings True Flexibility

Pradip Madan and Javed Iqbal
Peripheral Component Operations

Reprinted from the August 1981 edition of DATA COMMUNICATIONS magazine

ORDER NUMBER: 210499-001

Multiprotocol chip brings true flexibility

Pradip Madan and Javed Igbal, Intel Corporation, Santa Clara, Calif.

With this chip, a communications controller with two independent channels can handle a range of data rates as well as asynchronous, byte-synchronous, and bit-synchronous protocols.

In the beginning, data-handling machines were dedicated to one application at a time. As users developed a multiplicity of applications, manufacturers very quickly came through with new machines and peripherals fit for the chores. Consequently, today's data communications user has a plethora of communicating devices that do not necessarily communicate with each other. These machines often send and receive data at different speeds and operate with different protocols.

The usual method for achieving compatibility between different equipment is multiple serial-data input/output (I/O) boards (see "Evolution of data communications hardware"), which provide the user with a common speed and protocol between communicating devices. Such boards support only a limited speed range and a limited number of protocols. This inflexibility, which is a drawback in itself, has a major unfortunate consequence: These interfaces must be custom-designed and are thus expensive.

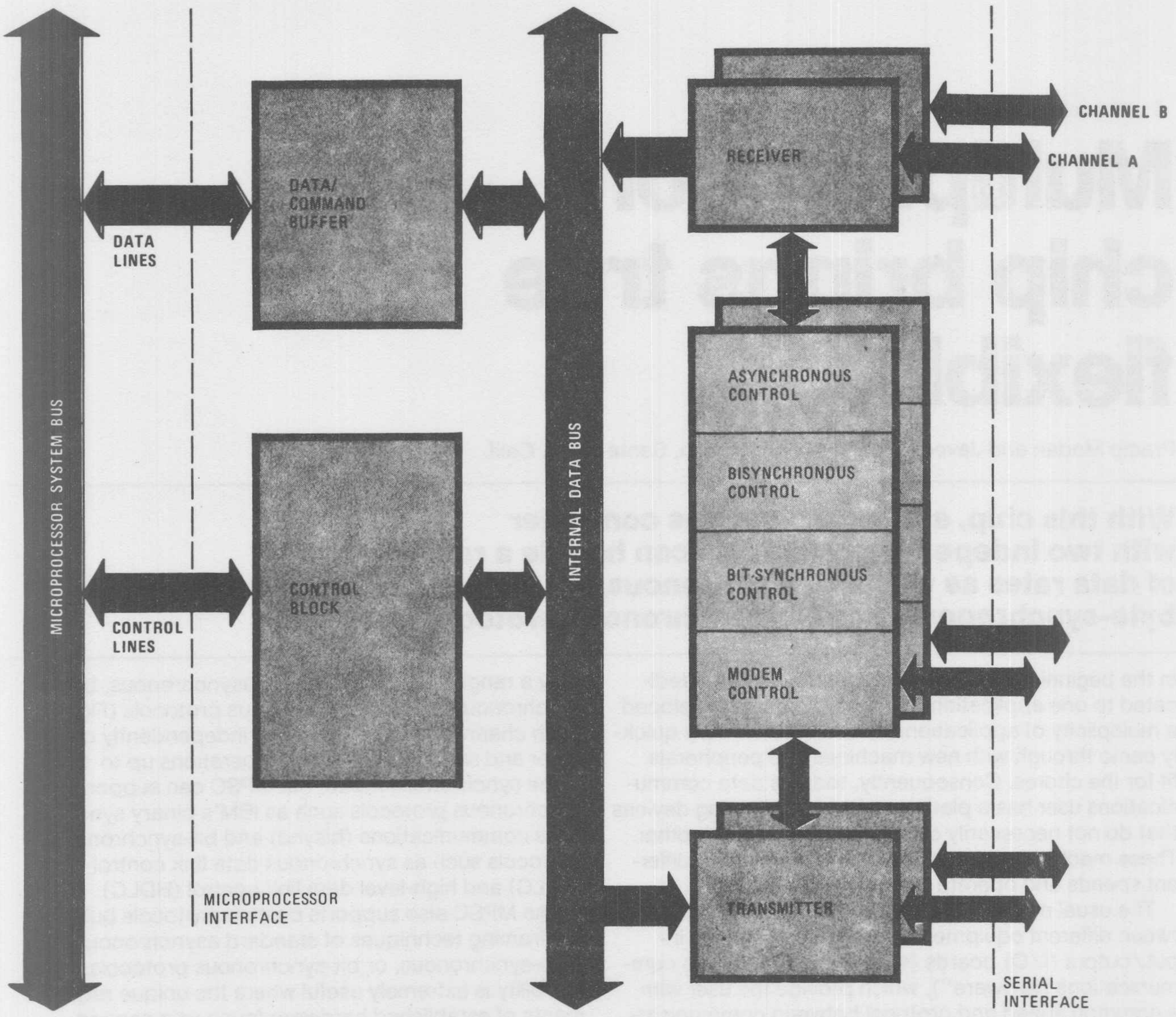
Intel Corporation of Santa Clara, Calif., has come up with an integrated circuit that can replace the expensive serial-data I/O boards. Called the multiprotocol serial controller, this one-chip data-communications microprocessor peripheral (dubbed the 8274 MPSC) can adapt to different data rates and protocols. It is being incorporated in several vendors' terminals, printers, statistical multiplexers, and computers, and its characteristics lend particular data communications features to this equipment.

The MPSC has two channels that can handle not only a range of speeds but also asynchronous, bit-synchronous, or byte-synchronous protocols (Fig. 1). Each channel can be configured independently of the other and supports full-duplex operations up to 1 MHz. In the synchronous mode, the MPSC can support byte-synchronous protocols such as IBM's binary synchronous communications (bisync) and bit-synchronous protocols such as synchronous data link control (SDLC) and high-level data link control (HDLC).

The MPSC also supports custom protocols built on the framing techniques of standard asynchronous, byte-synchronous, or bit-synchronous protocols. This flexibility is extremely useful where the unique requirements of established hardware (such as a second-generation mainframe) or special applications (such as a paper mill's control system) demand protocol customization (see "An overview of protocols").

For asynchronous communications, character format (Fig. 2A), parameters of length, number of stop bits, and parity are programmable. In most data communications equipment, these options are selected with the help of switches. The switch configuration allows a software routine to read their levels and program the MPSC accordingly. With the MPSC, 5 to 8 bits per character allow adaptation of a device to the code requirements of diverse applications, such as the 5-bit Baudot code used in amateur radio and older teletypewriters or the 7-bit ASCII or 8-bit EBCDIC character sets offered on electronic data processing (EDP) terminals. One, one and a half, and two stop bits allow

1. 8274 Multiprotocol serial controller. *Each of the two channels on the MPSC can handle not only a range of speeds but also asynchronous, byte-synchronous, or bit-synchronous protocols. Each channel can be configured independently of the other and, in addition, supports full-duplex operation.*



compatibility with high-speed electronic terminals or with teletypewriters that require a delay of up to two stop bits to strike and return the print hammer.

Handles errors too

The handling of error conditions is crucial to most computer users. The MPSC can detect character transmission errors via parity checking, using standard odd or even parity schemes. When character length is less than 8 bits, the receiver does not strip parity bits. They are either stored or passed through in a retransmitting environment, such as by a data concentrator that collects characters from its different links and transmits them with their parity bits in packet form. The receiver also detects overrun (received data not read before the next character is received) and framing-error conditions [missing stop bit(s)] and communicates them to the central processing unit (CPU) via the MPSC status register. Error recovery is determined by the equipment manufacturer's software.

Asynchronous data can be clocked at the transmitter or receiver clock rate or at 1/16th or 1/64th of that rate. For the receiver, the higher clock rate allows better synchronization of the sampling clock with the bit centers of the incoming data stream, guaranteeing relative immunity to clock skews. (Clock skews prevent data synchronization by slowing down or speeding up the clock in proportion to the data. They can occur on transmission lines because the frequency of the data waveform and hence its transmission speed may vary with the pattern of the bit stream.) The faster clock rate also guards against bit-boundary transients, which

are undefined waveform transitions when the bit changes from a binary 0 to a 1, or vice versa. Either condition, if it occurs, can cause the user to read the data incorrectly. In general, the faster the sampling clock rate, the better the synchronization and the smaller the chance of reading data while the transients are still rising or falling.

The need to send supervisory information, as well as data, can cause problems for users. For example, the transmitting station often needs to break off a transmission to send supervisory information. To do this, it may need to switch the link from the data to the diagnostic mode, or a user may press the Break key on a terminal to stop a program caught in an endless loop. Unfortunately, the Break condition is not standardized among manufacturers. One manufacturer may use character length as a time span for the Break condition, while another may use an absolute time span.

In MPSC, these differences are resolved through a simple mechanism for setting a Break for as long as desired. For example, a simple command written by the CPU to the MPSC will cause the data line to go low (Break condition) until another command is issued to end the Break (usually on releasing the Break key or after a timeout). At the receiver, the MPSC detects the occurrence or termination of Break and indicates it to the CPU via an interrupt. The CPU's action is determined by the equipment manufacturer's software. To prevent recognition of false starts (caused by glitches) or Break characters (data line low) at a remote receiver, the transmitter output is maintained at a high when the device is idle or initialized.

Byte-synchronous communications

In the byte-synchronous mode, the receiver starts its operation by hunting for sync character(s) (Fig. 2B) to establish synchronization with the transmitting station. Once it receives the sync characters, the receiver exits the hunt phase and starts assembling data bytes.

Although IBM bisync, a common byte-synchronous protocol, uses two sync characters, others use only one. A programmable option on the MPSC, which the user can invoke by command, allows synchronization to one or two 8-bit characters. For protocols requiring sync characters less than 8 bits long, synchronization can be achieved by external circuitry and indicated to the MPSC on an input pin. This input synchronizes the receiver with the incoming data stream.

Byte-synchronous protocols, like most message-oriented protocols, use some kind of block-check mechanism to detect transmission errors. For example, IBM uses cyclic redundancy check (CRC-16) in its bisync protocol and the SDLC polynomial in the SDLC protocol. Either of two error-checking polynomials—CRC-16 ($x16+x15+x2+1$) or SDLC ($x16+x12+x5+1$)—may be programmed for the MPSC's integrated CRC generator and checker. Ordinarily, CRC calculation takes a significant computational overhead both in CPU time and in data link throughput if done in software. Yet, the MPSC's integrated hardware CRC generation and checking requires less overhead and increases network performance.

Another exception-handling characteristic, the transmit-underrun feature, facilitates message termination without CPU intervention. An underrun occurs if data is not loaded into the transmitter before it becomes empty, which may be interpreted as the end of the message. The MPSC helps system throughput by taking the end-of-message monitoring burden from the CPU, informing the CPU of the underrun condition via an interrupt. Under the control of the equipment software, the MPSC can either flush out the CRC bytes followed by sync characters to terminate the message if there is no more data sent or send sync characters to maintain synchronization. Automatic CRC transmission is extremely useful for a character-count-oriented protocol such as Digital Data communications message protocol (DDCMP), common in Digital Equipment Corporation machines, since an accidental sync fill (unwarranted insertion of sync characters) invalidates the character count and the CRC calculation.

MPSC supports transparency to the control characters that determine various protocol functions and delineate message fields in byte-synchronous protocols by excluding particular characters—received or transmitted—from CRC accumulation. The CRC-accumulation enable/disable mechanism is operated under software control during the communications process.

Bit-synchronous communications

Bit-synchronous protocols (Fig. 2C) have become increasingly popular since they allow data to be transparent to the protocol. This enables flexibility of message length and bit pattern without the software overhead incurred by other transparency-supporting protocols, such as IBM bisync.

However, if the CPU must support all the bit-synchronous-protocol support functions in software, link performance can be severely limited. By relieving the CPU of several of the link-level bit-synchronous-protocol support functions, the MPSC improves throughput. It automatically inserts and deletes 0s after five consecutive 1s—except during a flag or abort sequence. It appends or deletes a frame's opening and closing flags and transmits or strips continuous flags when it is idle or enabled. It executes the frame-check-sequence calculation and verification using the SDLC CRC polynomial. All of these functions are basic to the principal bit-synchronous protocols SDLC and HDLC/ADCCP (advanced data-communications control procedure), as well as to their derivatives such as UDLC (universal data link control, used by Sperry-Univac) and BDLC (Burroughs data link control).

Although the message-framing technique is common among various bit-synchronous protocols, the address, control, and information-block formats differ. For example, SDLC specifies 8-bit addresses, while HDLC allows extended addressing. The MPSC transmits the address, control, and information fields under software control by writing them as consecutive bytes to the MPSC. If programmed for address search, the receiver can recognize its secondary address, as well as the universal address on the incoming data stream; other-

An overview of protocols

Asynchronous and synchronous. Bit and byte boundaries have to be recognized on an incoming data stream to permit character assembly. Bit synchronization allows the line to be sampled at the appropriate rate and times in order to sense binary 1s and 0s correctly. Byte synchronization enables the framing of serial data into characters. The lack of an accurate bit-synchronization signal from the transmitter to the receiver distinguishes asynchronous and synchronous communications. In asynchronous communications (Fig. 2A), the receiver clocks data from a local clock whose frequency is set approximately at the known transmitter frequency. Framing is accomplished when the receiver recognizes a start bit at the beginning and one or more stop bits at the end of each character.

Synchronous communications requires the transmission of a bit-synchronization signal, either encoded within the data stream or in parallel with it. Special synchronous characters that cannot be confused with any bit pattern in the data stream are used for byte-synchronization. Framing information for each character can be dispensed with, since the presence of a precise synchronization signal keeps the bit stream synchronized considerably longer than in asynchronous communications.

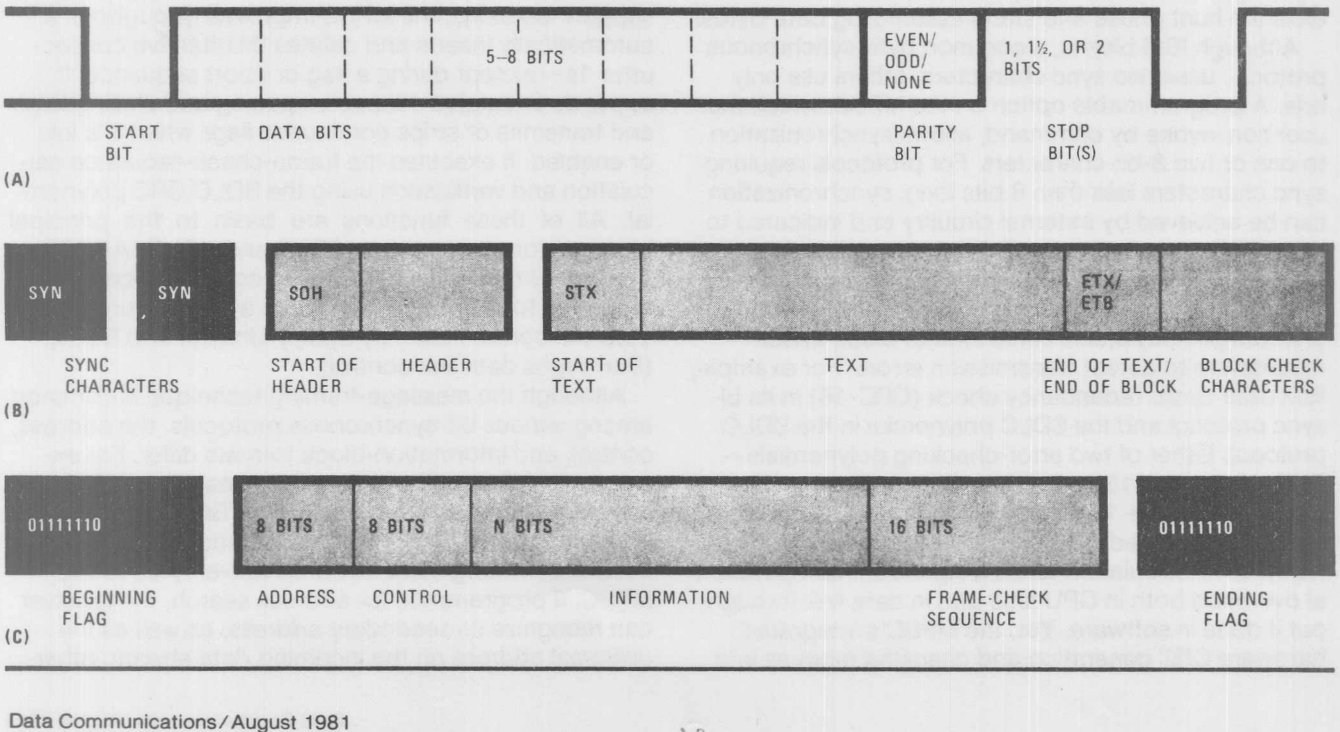
Byte- and bit-synchronous protocols. Although both byte- and bit-types of protocols use special synchronous characters at the beginning and end of a frame for byte-synchronization, the technique of formatting the frame distinguishes byte-synchronous protocols from bit-synchronous ones.

Byte-synchronous protocols use explicit frame-formatting information in the form of control characters to delineate the various message fields. For example, IBM's bisync protocol, shown in Figure 2B, uses special characters to indicate start-of-header, start-of-text, and end-of-text. These characters are predefined in popular character-set codes such as ASCII and EBCDIC. In addition, the bisync protocol requires the inclusion of cyclic-redundancy-check characters.

The use of special characters requires the receiving station to be able to identify them in the incoming data stream. Confusion may result from the incorrect recognition of a control bit pattern if it occurs unintentionally. In byte-synchronous protocols, transparency to the control characters is obtained either through character count or by prefixing data characters that have the control characters' bit pattern with another special control character.

Bit-synchronous protocols overcome the transparency problem by keeping the control information minimal and implicit. SDLC, for example (Fig. 2C), always requires an 8-bit address field, followed by an 8-bit control field. A variable-length information field follows the control field and is, in turn, followed by the 2-byte frame-check field. Beginning and ending synchronization flags—predefined to be 01111110—sandwich the message. To prevent the flags from being unintentionally detected within the message, the transmitter prevents the occurrence of the flag's bit pattern by inserting a 0 after each sequence of five 1s detected. At the destination, the receiver automatically deletes a 0 after every five consecutive 1s detected. This method is called zero-bit insertion and deletion.

2. Message formats. *In asynchronous mode (A), the receiver clocks data from a local clock. Start and stop bits frame the message. In bisync message format (B), special characters indicate start-of-header, start-of-text, and end-of-text. An SDLC protocol message (C) is framed by flags and requires an 8-bit address and control field.*



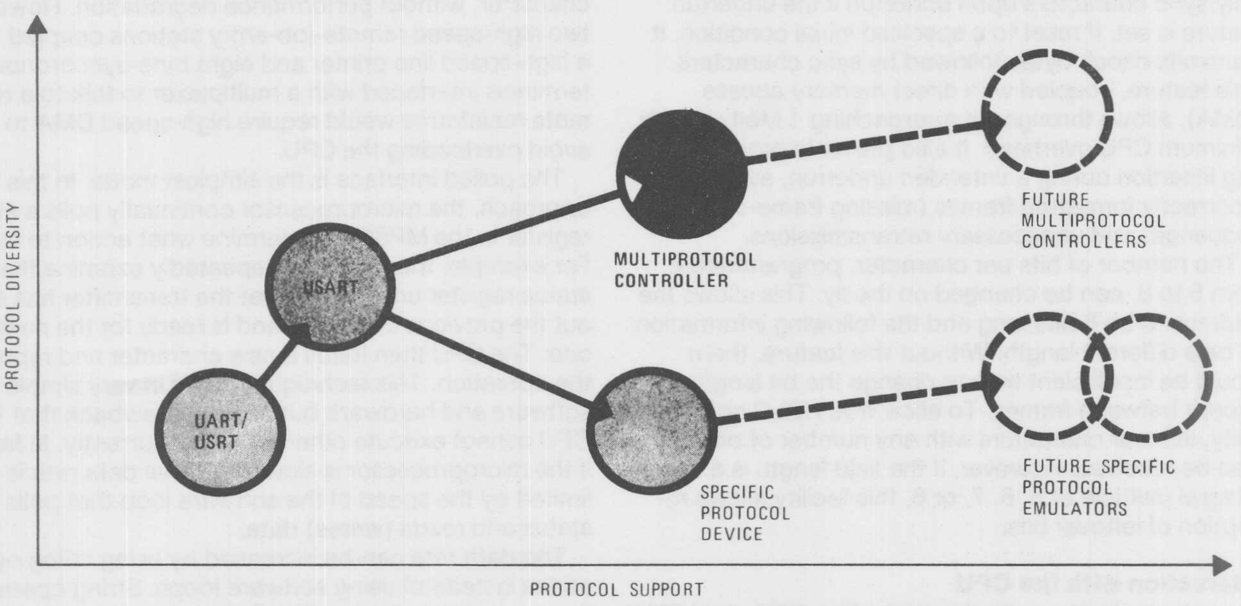
Evolution of data communications hardware

Only transceivers and level translators could be considered communications-oriented components in the discrete component and small-scale integration (SSI) days. These components merely satisfied the requirements of the physical layer of the communications link. Medium-scale integration (MSI) technology helped reduce the number of components by providing counters, I/O ports, shift registers, and comparators.

Large-scale integration (LSI) solutions, such as asynchronous/synchronous receiver/transmitter data communications elements, simplified the task of the electrical engineer considerably. Coupled with LSI, the arrival of the microprocessor shifted the remainder of the data link implementation burden to the programmer. As software came to be regarded as the best means of implementing different protocols, equipment manufacturers turned their focus on CPU intelligence and kept it there as the technology advanced.

However, specialized protocols, throughput requirements, and the cost of software development, maintenance, and execution (memory requirements) spurred the birth of a second generation of highly intelligent and specialized data communications components that executed specific protocols. Good examples of these specialized devices are Motorola's 6854 and Intel's 8273, which implement SDLC/HDLC.

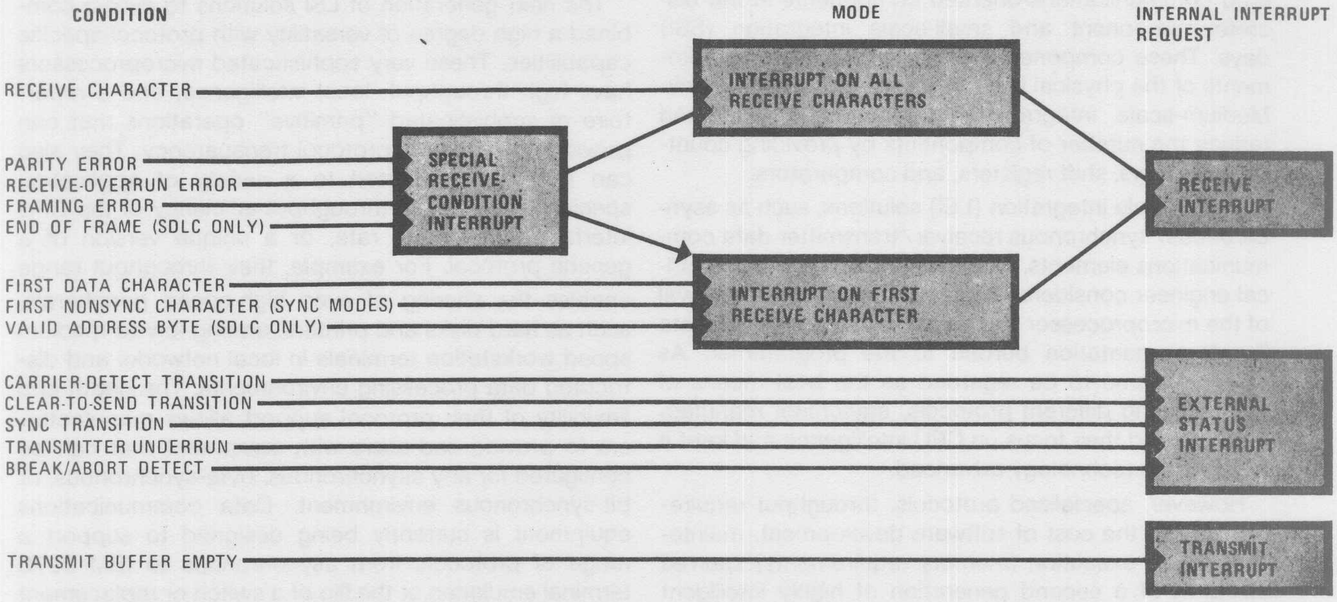
The next generation of LSI solutions to evolve combined a high degree of versatility with protocol-specific capabilities. These very sophisticated microprocessors have high throughput, local intelligence, and a repertoire of sophisticated "primitive" operations that can provide considerable protocol transparency. They also can be readily adapted to a variety of application specifics such as the throughput efficiency of the CPU interface, serial data rate, or a unique version of a generic protocol. For example, their throughput range enables the sharing of even high-speed peripherals, such as hard disks and printers, among low-to-medium speed workstation terminals in local networks and distributed data processing environments. The depth and flexibility of their protocol support allows manufacturers to provide end-users with equipment that can be configured for any asynchronous, byte-synchronous, or bit-synchronous environment. Data communications equipment is currently being designed to support a range of protocols from asynchronous to IBM 3270 terminal emulation at the flip of a switch or replacement of a ROM. The design, economy, and reliability of this equipment are facilitated by the new, still-evolving components.



wise, it receives all frames.

Extended addressing for HDLC is handled by the CPU. During address search, the MPSC transfers the second address byte to the CPU as a data character. If the address field is incorrect, the CPU sets the hunt mode, which suspends reception until a flag leading a new frame is received. This process is repeated until the bit pattern matching the extended address is

3. MPSC interrupts. *Receive, transmit, and external or status conditions are the three main sources of interrupts that exist for each channel, making a total of six interrupt groups. The receiver can interrupt the central processing unit on all characters or on the first character. A flexible configuration makes possible real-time responses.*



found. Control and information bytes are transparent to the MPSC and are transferred by it to the CPU for interpretation. Since the hunt mode is done in hardware instead of software, link throughput is increased.

As in the byte-synchronous mode, the MPSC sends only sync characters upon underrun if the underrun feature is set. If reset to a specified initial condition, it transmits check bytes followed by sync characters. This feature, coupled with direct memory access (DMA), allows throughput approaching 1 Mbit/s, with minimum CPU overhead. It also prevents premature flag insertion during unintended underrun, avoiding incorrectly formatted frames (missing frame-check sequence) and unnecessary retransmissions.

The number of bits per character, programmable from 5 to 8, can be changed on the fly. This allows the address to be 8 bits long and the following information to be a different length. Without this feature, there would be insufficient time to change the bit lengths except between frames. To allow true HDLC compatibility, leftover characters with any number of bits can also be received. However, if the field length is a nonintegral multiple of 5, 6, 7, or 8, this facility allows reception of leftover bits.

Interaction with the CPU

For communicating with the controlling CPU, the MPSC provides four programmable modes—polled, interrupt, DMA, and wait—as well as mode combinations such as DMA on Channel A and interrupt on Channel B. Polling is possible with all mode combinations. These options allow performance, hardware, and software trade-offs, enabling optimization to user requirements. Equipment costs, the data-transfer rate, and the non-data-communications tasks assigned to a CPU dictate the choice of the best interface option for the user. For example, the controlling CPU for a multiplexer that interfaces to four interactive CRTs through low-speed asynchronous lines can handle interrupts from four receiver and four transmitter sources, character by character, without performance degradation. However, two high-speed remote-job-entry stations coupled with a high-speed line printer and eight byte-synchronous terminals interfaced with a multiplexer to talk to a remote mainframe would require high-speed DMA to avoid overloading the CPU.

The polled interface is the simplest mode. In this approach, the microprocessor continually polls a status register in the MPSC to determine what action to take. For example, the CPU may repeatedly examine the status register until it sees that the transmitter has sent out the previous character and is ready for the next one. The CPU then loads a new character and repeats the operation. This technique results in very simple software and hardware but has the drawback that the CPU cannot execute other tasks concurrently. In fact, if the microprocessor is slow, the serial data rate is limited by the speed of the software loop that polls the status and reads (writes) data.

The data rate can be increased by using string operations instead of using software loops. String operations are slightly more efficient than looped ones since a single string instruction replaces a loop routine to repeatedly invoke the same operation. They also execute faster by economizing on instruction fetch (a single string instruction is fetched into the CPU only once, unlike a set of instructions repeated in a loop) and address calculation (each time the central processing unit fetches an instruction from memory, the instruction address must be calculated).

If data blocks are to be transferred, the ready lines—one for each channel—that signal a pin on the CPU that the MPSC is ready, can be used to synchronize CPU-controlled string I/O operations to the MPSC's throughput speed. The ready outputs, conditioned by the state of their respective transmitters and receivers, can be fed to the CPU ready input to "freeze" the CPU read or write cycles until the MPSC is ready for the next byte. While this mode does not allow the CPU to concurrently execute other tasks, it does increase the serial data throughput to the MPSC's throughput limit or the CPU's execution speed for the particular string instruction—whichever is slower. Both the string operation and the ready mode of operation, however, are faster than the software loop and give the user better throughput and memory-space utilization.

Wasted time waiting

Most computer users have experienced the computer's ceasing to respond to any commands while waiting for data input or output. Such behavior is typical of polled or ready-mode CPU interface. To enable the CPU to execute other tasks while waiting for I/O, the MPSC provides interrupt and DMA options.

In the interrupt mode, the CPU services the MPSC only when the MPSC is ready for I/O. The MPSC indicates readiness by interrupting the CPU in its current task. The CPU switches its control from the current task to a data communications service routine and returns to the original task on completion of the data communications task. Fourteen programmable interrupt sources per channel on the MPSC (Fig. 3) allow implementation of this concept. However, the hardware required for the interrupt mode is ordinarily more complex than that for the polled mode, with priority resolution (for devices competing for CPU time) and other circuits required to direct the CPU to the service routine appropriate to the interrupt source.

Transmit, receive, and external or status conditions are the three main sources of interrupts. In the MPSC, a flexible interrupt configuration can be optimized for the real-time response requirements of data communications equipment because of the method of assigning priorities to the six main interrupt groups (three per channel), internal priority-resolution circuitry, nonvectored and vectored interrupt modes (control is transferred to a service routine either by an external interrupt controller or by the MPSC), and the option of one MPSC's being able to block the interrupts of MPSCs with lower priorities until its own request is serviced.

An empty transmitter buffer is one condition that causes a transmit interrupt. The receiver can interrupt the CPU on all characters or on the "first character." The first character refers to the first data character in the asynchronous mode, the first message byte in the byte-synchronous mode, or a valid address byte in the bit-synchronous mode. This interrupt is useful in ready and DMA modes for initiating block transfers. Special receiver conditions, such as a parity error, framing/CRC error, buffer overrun, or SDLC/HDLC end-of-frame, also cause interrupts in both the first-character and the all-characters modes. External or status interrupts facilitate response to time-critical events such as loss of carrier or synchronization, transmitter underrun and the consequent sending of CRC bytes to prevent insertion of sync characters when no characters are being sent, and receipt of a break (asynchronous) or abort (bit-synchronous) sequence.

Character-by-character interrupts to the CPU impose a significant overhead on the CPU's time because of the context-switch overhead incurred (necessary to both save information about the interrupted task and get the information to start a new task). In comparison with this overhead, the actual character-transfer time may be small. Users of computer graphics are familiar with the manifestations of these overhead and throughput problems. For example, transmission of Picasso's Don Quixote would be completed intermittently because data is intermittently received from the remote CPU over the serial link. Each time the transmitting CPU stopped data I/O to attend to another task, the picador and horse in the picture would temporarily freeze in mid-construction at the receiving end. The solution to these overhead and throughput problems is to relieve the CPU of the data-transfer burden and allow data I/O and CPU activity in parallel. DMA-mode transfers, requiring a DMA controller, allow this. The DMA controller transfers data transparently to the CPU, except for initializing the controller at the beginning of a transfer or interpreting the data received. The CPU, therefore, has more time to devote to other tasks.

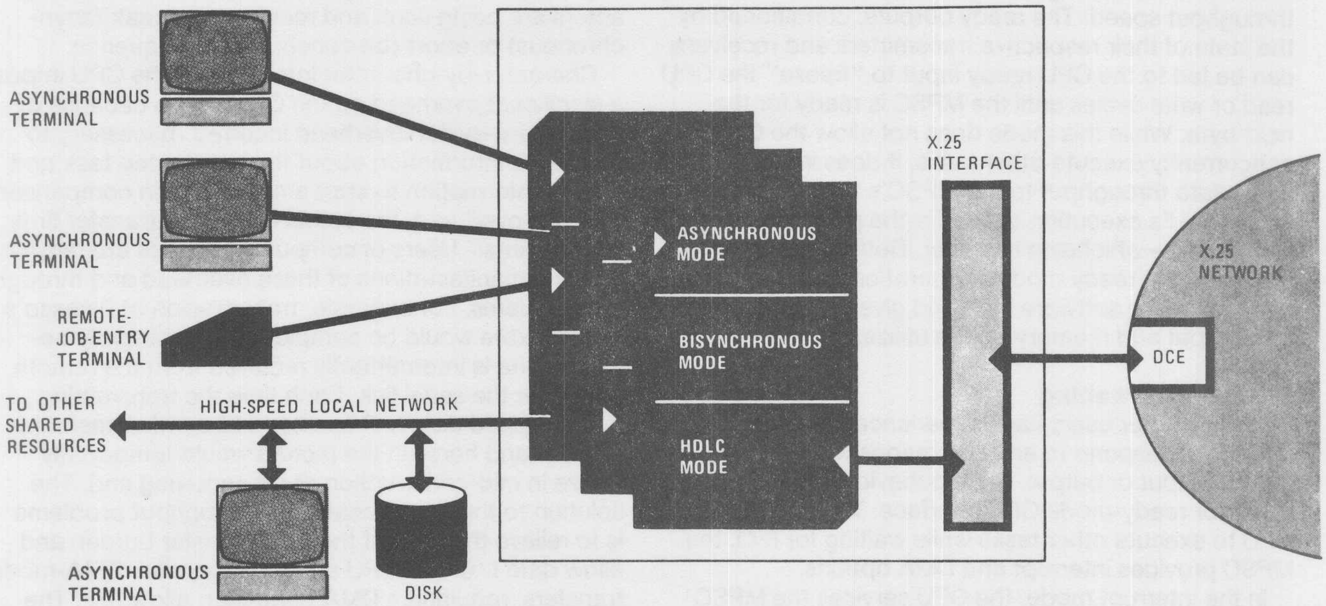
Modem interface

Four modem controls—two input, one output, and one input/output in only the asynchronous mode—are provided for each MPSC channel. Because of differing interface standards, variations within standards, and diverse handshaking needs among applications, modem controls are designed to be general. For example, the roles of modem controls are symmetrically opposed in data terminal versus data communications equipment, and the handshaking procedure with the communications devices differs among peripherals.

Clear-to-send (CTS) and carrier-detect (CD) levels may be programmed to interact with the transmitter and receiver logic of the MPSC. In the interacting mode, they enable the transmitter and receiver, respectively, facilitating direct control of the MPSC by the modem or destination peripheral rather than through CPU intervention. Since the signals control the MPSC directly, even more responsibility is offloaded from the CPU, thus increasing throughput.

In the noninteracting mode, these signals operate as general-purpose inputs that the CPU can read to make decisions about status changes occurring during the communications process. The noninteracting mode allows use of the MPSC in systems that interface to nonstandard equipment. Level changes in either direction on the modem control inputs cause interrupts and provide a time- and software-efficient alternative to CPU polling. The other two modem controls, data terminal ready (DTR) and request to send (RTS), are in the noninteracting mode and allow handshaking with the transmitter/receiver at the other end of the link.

__4. Assembling packets.__ Multiple MPSCs and the interfacing and support circuitry are used in an X.25 packet assembler/disassembler, which interconnects an X.25 network, several links with different protocols, and a high-speed local network. The complexity of the PAD depends on the interfacing devices.



It is difficult in almost any design to formulate goals to accommodate the user's unforeseen needs and to devise an optimum implementation strategy for these goals. However, data communications equipment is especially vulnerable to obsolescence from inadequate performance, flexibility, capacity, or reliability, since the components form critical links in the overall network operation. For example, a user who wants to add a new hard-disk controller to his workstation may be unable to use a spare channel available in his I/O subsystem, because the channel's slow speed causes an underutilization of the disk's capability, creating a bottleneck for the whole network. Alternatively, the addition of a bisync-compatible peripheral may preclude the use of asynchronous-only channels already available to the user. Or, the addition of terminals to a multiplexer may be limited by channel availability.

The data processing behind the communications
Consider the design of an X.25 packet assembler/disassembler (PAD). A PAD (Fig. 4) is, loosely speaking, a concentrator interfacing several links (carrying possibly divergent protocols at possibly divergent speeds) with one or more high-speed links. The PAD's complexity can vary depending on the number and types of devices it interfaces. The choice of the controller for the PAD would depend on the number of links, the data transfer speed over the links, the protocol requirements, the hardware and software overhead, and the flexibility and upgradability needed in the choice of these parameters. Speed, protocol, the number of channels, ease of implementation, flexibility, and upgradability would define the choice of the central processing unit as well.

The 8274 MPSC can be used with many CPUs and provides a fully compatible interface to a range of Intel microprocessors such as the 8048, 8049, 8051 (single-chip devices with memory); 8080, 8085 (8-bit microprocessors); 8086, 8088 (16-bit microprocessors); and the 8089 (an I/O processor). With appropriate circuitry, other manufacturers' microprocessors may be used. Through mode selection, these PAD interfaces can be further adapted to meet a node's throughput requirements or hardware availability.

Nodes that require cost-effective low-to-medium performance (50 to 100 kbit/s) and that need not be upgraded can be built around the conventional 8080/8085 CPUs. For example, a very simple PAD supporting two low-speed asynchronous and two bisync channels in the polling mode can be adequately served by a dedicated 8080 or 8085 CPU. Users who need a higher-performance, upgradable companion to the MPSC should use a microprocessor with pipelined architecture (one that uses parallel instead of serial processing) and an instruction set suited to fast program-execution time, such as the 8088.

To implement the link-level drivers, a supervisory program and perhaps even some user-level support for some of the peripherals right on the PAD itself would require a sizable amount of code. A microprocessor, like the 8088, with an architecture that lends itself to modular software and has a higher-level language orientation is a help in designing, implementing, and debugging the software, thus reducing development time and cost and generating reliable software.

Finally, at a still higher level of sophistication, the use of dedicated CPUs for very-high-speed interfaces (1 Mbit/s) to multidrop networks feeding into the PAD would require a multiprocessing architecture and could be supported with several 8088s. ■